Task 3 CPA

```
def g0(k):
    new_k = task1.g_calc(k)
    # print(new_k)
    return new_k[:len(k)]


def g1(k):
    new_k = task1.g_calc(k)
    # print(new_k)
    return new_k[len(new_k)//2:len(new_k)//2+len(k)]
```

G0

Function is called when the input string being iterated over encounters digit 0. It calls PRG created in task1 and from the bit string of length 2*x returned, it returns the first x bits to the calling function.

G1

Function is called when the input string being iterated over encounters digit 1. It calls PRG created in task1 and from the bit string of length 2*x returned, it returns the last x bits to the calling function.

```python
def encrypt(IV, key, message_blocks):
    encrypted_text = IV

    for i in range(len(message_blocks)):
        IV = bin(int(IV, 2) + 1).replace("0b", "")
        # print(IV)
        randomized_IV = task2.pseduoRandFunc(key, IV)
        # print(IV, randomized_IV)
        ci = xor(message_blocks[i], randomized_IV)
        # print(ci)
        encrypted_text += ci

    # print(encrypted_text)
    return encrypted_text
```

Encrypt()

Takes in Initialization Vector, Key, and Message_Blocks[] list as input. It calls PRF developed in task 2 to encrypt blocks along with using randomized counter mode as the mode of operation. It finally returns an encrypted text with length IV + message_len as output.

```python
def decrypt(IV_len, key, cipher_text):
    IV = cipher_text[:IV_len]
    cipher_text = cipher_text[IV_len:]

    no_of_blocks = math.ceil(len(cipher_text) / IV_len)
    n = IV_len

    message_lis = []
    for i in range(no_of_blocks):
        mi = cipher_text[i*n:(i+1)*n]
        message_lis.append(mi)

    plain_text = ""

    for i in range(len(message_lis)):
        IV = bin(int(IV, 2) + 1).replace("0b", "")
        randomized_IV = task2.pseduoRandFunc(key, IV)
        pi = xor(message_lis[i], randomized_IV)
        plain_text += pi

    return plain_text
```

Decrypt()

Decryption function on the receiver's end, it takes IV as a parameter. Key and the cipher_text. We assume that the receiver has its key already. Besides that the r (here IV) along with the cipher_text is sent from the sender's side. The decrypt function deciphers the message block by block and appends the output of various blocks to finally have the message string as plain_text which it returns.

```python
def cbc_prf(IV, key, message):
    n = len(IV)
    no_of_blocks = math.ceil(len(message) / n)

    message_lis = []
    # print(message)
    for i in range(no_of_blocks):
        mi = message[i*n:(i+1)*n]
        message_lis.append(mi)

    cipher_text = encrypt(IV, key, message_lis)
    actual_text = decrypt(n, key, cipher_text)

    print("Encrypted: ", cipher_text)
    print("Decrypted: ", actual_text)
```

Cbc_prf()

The driver function of the task. It receives IV, key and message from the user as input. It breaks the message into blocks of length similar to length of IV, calls the respective methods as required.

Sample Input

IV: 10001

Key: 10110

Message: 11101010111110101010101010

Sample Output

Encrypted: 10001111010101011101011101010100

Decrypted: 11101010111110101010101010